

# Adaptively Restrained Molecular Dynamics in LAMMPS

Krishna Kant Singh<sup>1,2,3</sup> and Stephane Redon<sup>1,2,3</sup>

<sup>1</sup>NANO-D, INRIA

<sup>2</sup>Univ. Grenoble Alpes, LJK, F-38000 Grenoble, France

<sup>3</sup> CNRS, LJK, F-38000 Grenoble, France

E-mail: `stephane.redon@inria.fr`

**Abstract.** Adaptively Restrained Molecular Dynamics (ARMD) is a recently introduced particles simulation method that switches positional degrees of freedom on and off during simulation in order to speed up calculations. In the NVE ensemble, ARMD allows users to trade between precision and speed while, in the NVT ensemble, it makes it possible to compute statistical averages faster. Despite the conceptual simplicity of the approach, however, integrating it in existing molecular dynamics packages is non-trivial, in particular since implemented potentials should a priori be rewritten to take advantage of frozen particles and achieve a speed-up. In this paper, we present novel algorithms for integrating ARMD in LAMMPS, a popular multi-purpose molecular simulation package. In particular, we demonstrate how to enable ARMD in LAMMPS without having to re-implement all available force fields. The proposed algorithms are assessed on four different benchmarks, and show how they allow us to speed up simulations up to one order of magnitude.

*Keywords:* Algorithms, Adaptively Restrained Molecular Dynamics, Incremental force computation, LAMMPS, Molecular Dynamics, Neighbor List

Submitted to: *Modelling Simul. Mater. Sci. Eng.*

## 1. Introduction

Molecular Dynamics (MD) simulations are widely used to understand complex systems, including *e.g.* liquids, solids, biomolecules, etc. [1, 2, 3, 4]. By providing the positions of particles as a function of time, in particular, MD simulations help rationalize the behavior of complex systems [5, 6, 7, 8, 9]. Because MD involves integrating Newton’s equations of motion using discrete time steps, however, and because time steps sizes are typically small (for biological systems, for example, the time step size is usually a few femtoseconds), MD has difficulty simulating the timescales over which many important processes take place (like protein folding, rare events etc.) [10, 11].

Numerous attempts have been made to accelerate MD simulations, for example by increasing the time step size or through the use of multiple time steps [12, 13], by adding an external biased potential (metadynamics, umbrella sampling, etc.) [10, 11], appropriately reducing the accuracy of the simulation (coarse-grain simulations and multi-scale simulations) [14, 15, 16], and even building special-purpose supercomputers for performing MD [17].

Adaptively Restrained Molecular Dynamics (ARMD) is a recent approach that attempts to tackle the timescale issue by reducing the number of computations per time step [18]. In ARMD, particles adaptively switch their positional degrees of freedom on and off during the simulation, based on their instantaneous kinetic energy. Precisely, particles whose kinetic energy is sufficiently large are considered *active*, and have normal dynamics, while particles whose kinetic energy is below some threshold are *restrained*, and stop moving completely. The status of particles evolve during simulation, and the Adaptively Restrained Hamiltonian ensures that stable simulations can be performed, and that statistics can be recovered [18]. Since inter-atomic forces typically depend upon relative particle positions, only forces involving active particles need to be updated at each time step [19]. This may result in significant speed-ups, depending on the chosen simplification thresholds.

A typical MD step entails repetition of the following steps:

- (i) Calculate the forces applied on particles
- (ii) Update the particles momenta
- (iii) Update the particles position

In these steps, the most intensive step is, by far, the force calculation. Inter-atomic forces can be divided into two main classes: bonded and non-bonded forces. Bonded forces include *e.g.* bonds, angles, and torsions. Non-bonded forces can be further divided into two classes: long- and short-range forces. Long-range interactions are often computed through Ewald summations [20, 21], while short-range interactions are typically truncated (*i.e.* vanish after a specific *cutoff distance*  $r_c$ ), and computed thanks to *neighbor lists*, *i.e.* lists of neighboring particles. These neighbor lists are often build through a combination of *cell lists* and *Verlet lists*[22]. In the cell lists method, all particles are binned into 3D cells of side length  $l \geq r_c$  according to their

coordinates. Neighbors of particles in a given cell are only searched in the cell itself and its 26 neighboring cells (instead of the whole simulation box). In the Verlet lists method, each particle is associated to a list of neighboring particles within distance  $r_s = r_c + \delta$ , where  $\delta$  is a *buffering distance*. This list is updated either every  $N$  time steps, or based upon how far the particles have moved [23, 24]. Some short-range force calculations also use Newton’s third law and store only half of the neighbor pairs.

Despite the conceptual simplicity of the ARMD approach, however, only simple implementations have been demonstrated so far [18, 25], and integrating it in an existing molecular dynamics package (*e.g.* LAAMPS, GROMACS, etc.) is non-trivial. Indeed, the potentials implemented in these simulation packages should take into account the fact that some particles are frozen in order to speed up force calculations [19], and this would *a priori* require re-implementing all potentials.

In this paper, we present novel algorithms for integrating ARMD in LAMMPS, a popular multi-purpose molecular simulation package, *without* having to modify the available force fields. In particular, we introduce a novel method to compute neighbor lists and short-range forces when performing ARMD simulations. This method is independent of the underlying potential or force field, and may be used with any pair potential. We validate these new algorithms by simulating several systems in the NVE and NVT ensembles with the adaptively restrained version of LAMMPS. We show that our algorithms, combined with the AR molecular dynamics methodology, makes it possible to finely trade between precision and computational cost (in the NVE ensemble), and speed up the calculation of statistical properties (in the NVT ensemble).

## 2. Methods

### 2.1. Adaptively Restrained Molecular Dynamics

For completeness, we provide a brief overview of the ARMD methodology. For more details, we refer the reader to [18].

The time evolution of the system containing  $N$  particles may be derived from the Hamiltonian function 1

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2} \mathbf{p}^T \mathbf{M}^{-1} \mathbf{p} + V(\mathbf{q}) \quad (1)$$

where  $\mathbf{q}$  is a  $3N$ -dimensional vector of coordinates,  $\mathbf{p}$  is a  $3N$ -dimensional vector of momenta,  $\mathbf{M}$  is a  $3N \times 3N$  mass matrix, and  $V(\mathbf{q})$  is the potential energy.

In Adaptively Restrained (AR) molecular dynamics, the  $3N \times 3N$  inverse mass matrix  $\mathbf{M}^{-1}$  is replaced by a  $3N \times 3N$  inverse inertia matrix  $\Phi(q, p)$  which adaptively enforces restraints during simulation [18]:

$$H_{AR}(\mathbf{q}, \mathbf{p}) = \frac{1}{2} \mathbf{p}^T \Phi(\mathbf{q}, \mathbf{p}) \mathbf{p} + V(\mathbf{q}). \quad (2)$$

One possible choice for the inverse inertia matrix  $\Phi(\mathbf{q}, \mathbf{p})$  is a block-diagonal matrix  $\text{diag}[\Phi_1(\mathbf{q}_1, \mathbf{p}_1), \dots, \Phi_N(\mathbf{q}_N, \mathbf{p}_N)]$  with

$$\Phi_i(\mathbf{q}_i, \mathbf{p}_i) = m_i^{-1} [1 - \rho_i(\mathbf{p}_i)] \mathbf{I}_{3 \times 3} \quad (3)$$

where  $\mathbf{I}_{3 \times 3}$  is the  $3 \times 3$  identity matrix,  $m_i$ ,  $\mathbf{q}_i$  and  $\mathbf{p}_i$  are respectively the mass, position and momentum of particle  $i$ , and  $\rho_i$  is its *restraining function*:

$$\rho_i(\mathbf{p}_i) = \begin{cases} 1 & \text{if } 0 \leq K_i(\mathbf{p}_i) \leq \varepsilon_i^r \\ 0 & \text{if } K_i(\mathbf{p}_i) \geq \varepsilon_i^f \\ s(K_i(\mathbf{p}_i)) \in [0, 1] & \text{elsewhere} \end{cases} \quad (4)$$

In this definition,  $K_i(\mathbf{p}_i) = \frac{1}{2}m_i^{-1}\mathbf{p}_i^T\mathbf{p}_i$  is the kinetic energy of particle  $i$ ,  $\varepsilon_i^r$  is its *restrained-dynamics threshold*,  $\varepsilon_i^f$  is its *full-dynamics threshold*, and  $s$  is a  $\mathcal{C}^2$  function that smoothly interpolates between 0 and 1.

When  $\rho_i = 0$ ,  $\Phi_i = m_i^{-1}$  and particle  $i$  is *active* (the particle mass is unchanged). When  $\rho_i = 1$ ,  $\Phi_i = 0$  and particle  $i$  is *restrained*, *i.e.* will not move whichever force is applied to it (the particle mass is infinite). When  $\rho_i \in (0, 1)$ , particle  $i$  is in a *transition state*. The restraining function  $\rho_i$  above depends upon the kinetic energy of the particle. Precisely, particle  $i$  is restrained when its kinetic energy is smaller than the restrained-dynamics threshold ( $\varepsilon_i^r$ ), and it is active if its kinetic energy is larger than its full-dynamics threshold  $\varepsilon_i^f$ . Particles with kinetic energies between  $\varepsilon_i^r$  and  $\varepsilon_i^f$  are considered as transition particles.

To integrate the equations of motion of a system in the NVE ensemble using the AR Hamiltonian, we may use a modified Velocity Verlet algorithm which takes into account the non-constant mass matrix:

- (i)  $\mathbf{p}_i(t + \frac{1}{2}\Delta t) = \mathbf{p}_i(t) + \frac{1}{2}\mathbf{f}_i(t)\Delta t$
- (ii)  $\mathbf{q}_i(t + \Delta t) = \mathbf{q}_i(t) + \nabla_{\mathbf{p}_i}H_{AR}(t + \frac{1}{2}\Delta t)\Delta t$
- (iii)  $\mathbf{f}_i(t + \Delta t) = -\nabla_{\mathbf{q}_i}V(\mathbf{q}(t + \Delta t))$
- (iv)  $\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t + \frac{1}{2}\Delta t) + \frac{1}{2}\mathbf{f}_i(t + \Delta t)\Delta t$

To perform simulation in the canonical ensemble (NVT), we use AR Langevin dynamics [18, 26]:

$$\begin{aligned} d\mathbf{q} &= \nabla_{\mathbf{p}}H_{AR}(\mathbf{q}, \mathbf{p})dt \\ d\mathbf{p} &= -\nabla_{\mathbf{q}}H_{AR}(\mathbf{q}, \mathbf{p})dt - \gamma\nabla_{\mathbf{p}}H_{AR}(\mathbf{q}, \mathbf{p})dt + \sqrt{\frac{2\gamma}{\beta}}d\mathbf{W} \end{aligned} \quad (5)$$

where  $dW_t$  is a  $3N$ -dimensional Brownian motion and  $\gamma > 0$  is the frictional constant. Discretization of the modified Langevin is done using second-order Trotter splitting [26]. The temperature of the AR system is given by:

$$T = \frac{1}{DK_B} \left\langle \sum_{i=1}^N \left( \mathbf{p}_i \cdot \frac{\partial H_{AR}}{\partial \mathbf{p}_i} \right) \right\rangle \quad (6)$$

where  $D$  is the number of degrees of freedom in the system and the dot represents the dot product. In the version described above, the AR Hamiltonian is separable and the temperature is unchanged [18].

## 2.2. Molecular dynamics in LAMMPS

After setting up initial conditions, LAMMPS repeats the following steps:

---

**Algorithm 1:** LAMMPS integration step

---

```

1 if (UpdateNeeded) then
2   Build neighbor lists
3 end
4 Calculate forces
5 Update momenta
6 Update positions

```

---

where lines 1 – 2 ensure that neighbor lists are rebuilt periodically (*e.g.* every 20 time steps), and line 3 is the main step: calculating forces.

To calculate forces, LAMMPS provides force fields implementations with a list  $L$  of particles for which forces should be computed, as well as the neighbor list  $NL(i)$  of each particle  $i$  in  $L$ . LAMMPS may provide either *full neighbor lists* (FNLs) or *half neighbor lists* (HNLs). In the FNL case, all neighbors of particle  $i$  are stored in  $NL(i)$ . In the HNL case, the neighbor pair  $(i, j)$  is stored in either  $NL(i)$  or  $NL(j)$ . The HNL case is used when the force calculation step may use Newton’s third law ( $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$ ) in order to reduce the number of force calculations by half and speed up the simulation.

The force calculation algorithm is then:

---

**Algorithm 2:** *ComputeForces*( $L, NL$ )

---

```

1 for ( $i \in L$ ) do
2   for ( $j \in NL(i)$ ) do
3      $\mathbf{f}_{ij} \leftarrow \text{ComputeForce}(i, j)$ 
4      $\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{f}_{ij}$ 
5     if (NewtonOn) then
6        $\mathbf{f}_j \leftarrow \mathbf{f}_j - \mathbf{f}_{ij}$ 
7     end
8   end
9 end

```

---

where  $\mathbf{f}_{ij}$  is the force applied to  $i$  by  $j$ ,  $\mathbf{f}_i$  is the total force applied to  $i$ , and *NewtonOn* is true when HNLs and Newton’s third law are used.

## 2.3. Adaptively Restrained Molecular Dynamics in LAMMPS

In AR molecular dynamics, a system of  $N$  particles can be represented as a combination of  $N_R$  restrained particles and  $N_A = N - N_R$  active or transitioning particles. At any time-step, interactions between particles (*i.e.* inter-particle energies and forces) may thus be categorized into two types:

- (i) **Restrained interactions:** interactions between restrained particles only.
- (ii) **Active interactions:** interactions involving at least one active particle.

In most force fields, interactions only depend on *relative particle positions*. As a result, at any time step, restrained interactions do not have to be updated, and only active interactions must be recalculated (since active particles may have moved since the previous time step). An ARMD integration step may thus rely on the same general steps as in Algorithm 1, but efficient ARMD simulations require *incremental force calculation algorithms*, *i.e.* force calculation algorithms that only update active interactions.

In order to enable AR molecular dynamics in LAMMPS without modifying the source code of all force fields, our strategy is a) to pass modified information to force fields implementations, unbeknownst to them, so that they *only compute active interactions*, and b) to use these active interactions to incrementally update the total forces applied to particles (which are required to update positions and momenta). We note that modifying LAMMPS in such a way allows us to enable ARMD for force fields that are *not yet implemented*, which is a significant advantage of the approach proposed in this paper.

## 2.4. Active neighbor lists

Let  $C$  denote the complete list of particles,  $A$  denote the list of active (or transitioning) particles, and  $R$  denote the list of restrained particles, so that  $C = A \uplus R$ .

Since we want force fields implementations to compute active interactions only, we are going to pass them the list  $A$  of active particles instead of the complete list  $C$ . Assuming we may use Newton’s third law, the list  $A$  is sufficient, since any force  $\mathbf{f}_{ij}$  we need to compute involves at least one active particle. However, we cannot just pass to force fields the HNLs of active particles, since there would be a risk that these HNLs do not contain all the neighbors for which we need to compute interactions<sup>‡</sup>. Conversely, we should not pass the FNLs of active particles if we want to take advantage of Newton’s third law. As a result, we introduce *active neighbor lists* (ANLs), *i.e.* neighbors lists that are built in such a way that, when  $i$  is an active particle,  $ANL(i)$  contains  $j$  if a)  $j$  is restrained or b) if  $j$  is active and  $ANL(j)$  does not contain  $i$ . If  $i$  is restrained,  $ANL(i)$  is empty. The ANLs can thus be seen as HNLs for active-active neighbors (if both  $i$  and  $j$  are active, either  $ANL(i)$  contains  $j$  or  $ANL(j)$  contains  $i$ ), and FNLs for active-restrained neighbors (if  $i$  is active,  $ANL(i)$  contains all the restrained neighbors of  $i$ ). We may thus use the following algorithm to build the ANLs:

When we initialize the simulation (before the first time step), we first build the ANLs from the FNLs thanks to Algorithm 3. During the main loop, however, we

<sup>‡</sup> Consider *e.g.* the case of four particles  $1, \dots, 4$  that are all neighbors to each other, where  $HNL(1) = \{2, 3, 4\}$ ,  $HNL(2) = \{3, 4\}$ ,  $HNL(3) = \{4\}$  and  $HNL(4) = \emptyset$ , and assume that particles 1 and 4 are active. If a force field implementation only receives  $HNL(1)$  and  $HNL(4)$ , it will only compute active interactions  $\mathbf{f}_{12} = -\mathbf{f}_{21}$ ,  $\mathbf{f}_{13} = -\mathbf{f}_{31}$ , and  $\mathbf{f}_{14} = -\mathbf{f}_{41}$  (thanks to  $HNL(1)$ ), and will compute neither  $\mathbf{f}_{42} = -\mathbf{f}_{24}$  nor  $\mathbf{f}_{43} = -\mathbf{f}_{34}$ , since neither  $HNL(1)$  nor  $HNL(4)$  signal that particles 2 and 3 are neighbors of particle 4.

---

**Algorithm 3:** *BuildANL(i)*

---

```

1 for ( $j \in FNL(i)$ ) do
2   if ( $j \in R$ ) or ( $i \notin ANL(j)$ ) then
3      $ANL(i) \leftarrow ANL(i) \cup j$ 
4   end
5 end

```

---

incrementally update the ANLs based on the list  $S_A$  of particles switching to an active (or transitioning) state, and the list  $S_R$  of particles switching to a restrained state. Precisely, if  $A$  and  $R$  represent the state distribution of the simulation at time step  $n$ , then  $S_A$  and  $S_R$  are the lists of particles switching between time steps  $n$  and  $n + 1$ . We update the ANLs in two steps:

*Step 1: clearing ANLs of particles that become restrained*

When a particle  $i$  switches from active to restrained, we need to empty  $ANL(i)$ . However, we need to retain interactions with any neighboring particle  $j$  that remains active, *i.e.* insert  $i$  in  $ANL(j)$ . In the first step of the ANL lists update, we thus go through each particle  $i$  in  $S_R$  and clear  $ANL(i)$  using Algorithm 4.

---

**Algorithm 4:** *ClearANL(i)*

---

```

1 for ( $j \in ANL(i)$ ) do
2   if ( $j \in A$ ) and ( $j \notin S$ ) then
3      $ANL(j) \leftarrow ANL(j) \cup i$ 
4   end
5 end
6  $ANL(i) \leftarrow \emptyset$ 

```

---

*Step 2: building ANLs of particles that become active*

When a particle  $i$  switches from restrained to active, we need to build  $ANL(i)$ . In the second step of the ANL lists update, we thus go through each particle  $i$  in  $S_A$  and build  $ANL(i)$  using Algorithm 3.

*2.5. Incremental force updates*

As noted before, in order to achieve a speed-up through AR molecular dynamics, we need to be able to *incrementally* update the total forces applied on particles, *i.e.* avoid recomputing all total forces.

Let  $I$  denote the list of *involved particles*, *i.e.* the list of particles involved in active interactions (either because they are active particles, or because they are neighbors of active particles):  $I = A \cup (\cup_{i \in A} ANL(i))$ . The list  $I$  may be incrementally updated at each time step while updating the ANLs.

We may use  $A$  and the ANLs to incrementally update the total forces of particles in  $I$ :

- (i) Compute forces  $\mathbf{f}_{old}^+$  between particles in  $I$  using current positions.
- (ii) Subtract  $\mathbf{f}_{old}^+$  from the total forces  $\mathbf{f}$ .
- (iii) Update the positions of active particles.
- (iv) Compute forces  $\mathbf{f}_{new}^+$  between particles in  $I$  using the new positions.
- (v) Add  $\mathbf{f}_{new}^+$  to the total forces  $\mathbf{f}$ .

Algorithm 5 shows the pseudo-code of the initialization step (before the first time step), and Algorithm 6 gives the pseudo-code for performing an AR molecular dynamics step in LAMMPS.

---

**Algorithm 5:** AR-LAMMPS initialization step

---

```

1 for ( $i \in L$ ) do
2    $\mathbf{f}_i \leftarrow 0$ 
3    $\mathbf{f}_i^+ \leftarrow 0$ 
4    $ANL(i) \leftarrow \emptyset$ 
5 end
6 Construct all FNLs
7  $\mathbf{f} \leftarrow ComputeForces(C, FNL)$ 
8 Build  $A, R, I$ 
9 for ( $i \in A$ ) do
10    $BuildANL(i)$ 
11 end
```

---

### 3. Results and discussion

In order to validate our new algorithms and their implementation in LAMMPS, we performed ARMD simulations on a set of toy models that were general enough to model typical simulations, while simple enough to allow for detailed analysis. For some benchmarks, in particular, we chose test cases that were similar to earlier ones [18] to demonstrate that speed-ups were still achievable despite computational overheads that may result from integration in a MD package not initially designed for adaptively restrained simulation.



---

**Algorithm 6:** AR-LAMMPS integration step

---

```

1 Update momenta
2 Update  $A$ ,  $R$ 
3 if ( $UpdateNeeded$ ) then
4   Construct all FNLs and empty all ANLs
5   for ( $i \in A$ ) do
6      $BuildANL(i)$ 
7   end
8 end
9 else
10   Build  $S_A$  and  $S_R$ 
11   for ( $i \in S_R$ ) do
12      $ClearANL(i)$ 
13   end
14   for ( $i \in S_A$ ) do
15      $BuildANL(i)$ 
16   end
17 end
18 Update  $I$ 
19  $\mathbf{f}^+ \leftarrow ComputeForces(A, ANL)$ 
20 for  $i \in I$  do
21    $\mathbf{f}_i \leftarrow \mathbf{f}_i - \mathbf{f}_i^+$ 
22    $\mathbf{f}_i^+ \leftarrow 0$ 
23 end
24 Update positions
25  $\mathbf{f}^+ \leftarrow ComputeForces(A, ANL)$ 
26 for  $i \in I$  do
27    $\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{f}_i^+$ 
28    $\mathbf{f}_i^+ \leftarrow 0$ 
29 end

```

---

### 3.1. Systems of Lennard-Jones particles

We simulated three systems with different numbers (500, 4,000 and 108,000) of Lennard-Jones particles using an AR integrator in the NVE ensemble. All simulations were performed in reduced, Lennard-Jones units using our version of LAMMPS. Particles were generated along a fcc lattice of density 0.8442, and initial velocities were assigned according to the Boltzmann distribution. For all simulations, we used a time-step of 0.005. Interactions beyond distance  $2.5\sigma$  were ignored. Periodic boundary conditions were applied in all three directions. We chose values of  $\epsilon_r$  and  $\epsilon_f$  in order to achieve specific ratios of restrained particles.

We ran one reference simulation and one ARMD simulation for all three systems. Reference simulations were performed with the original LAMMPS neighbor list and force update algorithms, whereas ARMD used ANLs and incremental force update algorithms. To determine the resulting speed-ups, we compared the average times spent in each integration steps. Figure 1 shows the achieved speed-ups with respect to the percentage of restrained particles in the system. Figure 1 shows that, in order to have a speed-up for systems containing 500 and 4000 particles, at least 60% of the particles should be restrained. A 1.6X to 2.8X speed-up was observed when 80% to 90% particles were restrained. For the system containing 108,000 particles, we observed a speed-up when at least 50% of the particles were restrained, and a 2.5X to 4.2X speed-up was observed when 80% to 90% of the particles were restrained. One reason for achieving higher speed-ups in larger systems is the number of force calculations performed per time step. Smaller systems contain relatively fewer force calculations per time-step, and reducing them does not have much influence on the overall speed-up. For larger systems, force calculations constitute the major part of the time-step cost, and reducing them can significantly speed-up the simulation. Figure 2 shows that, even for AR simulations, the total adaptive energy of the system is constant<sup>§</sup>.

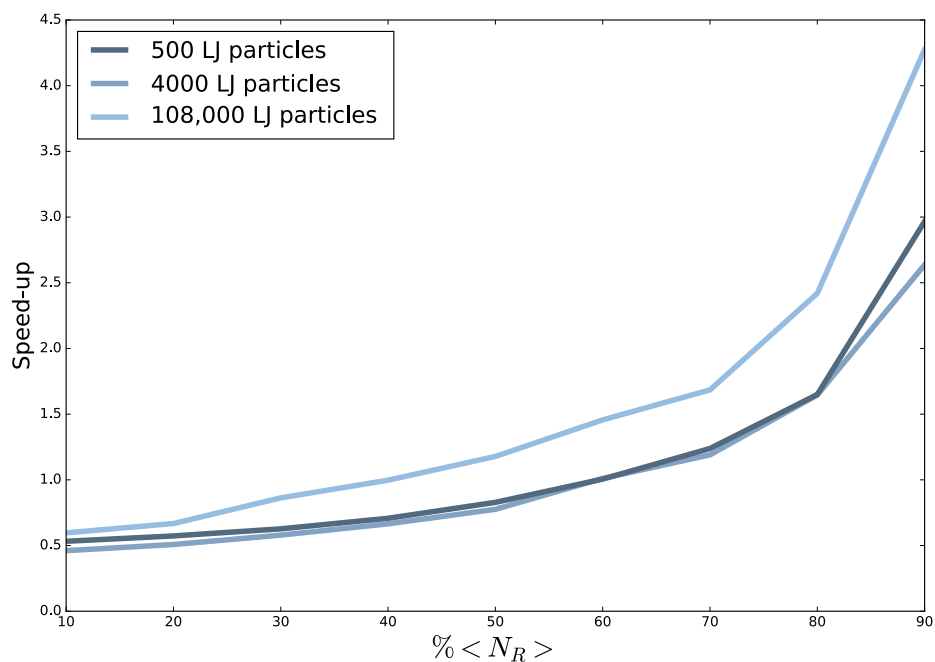
In order to study the structural properties of the system, we also performed NVT simulations of all three systems using two different sets of AR parameters. We computed the radial distribution function (RDF) of the systems using classical MD and ARMD. Figure 3 shows that the RDFs obtained by both methods coincide.

### 3.2. Collision cascade

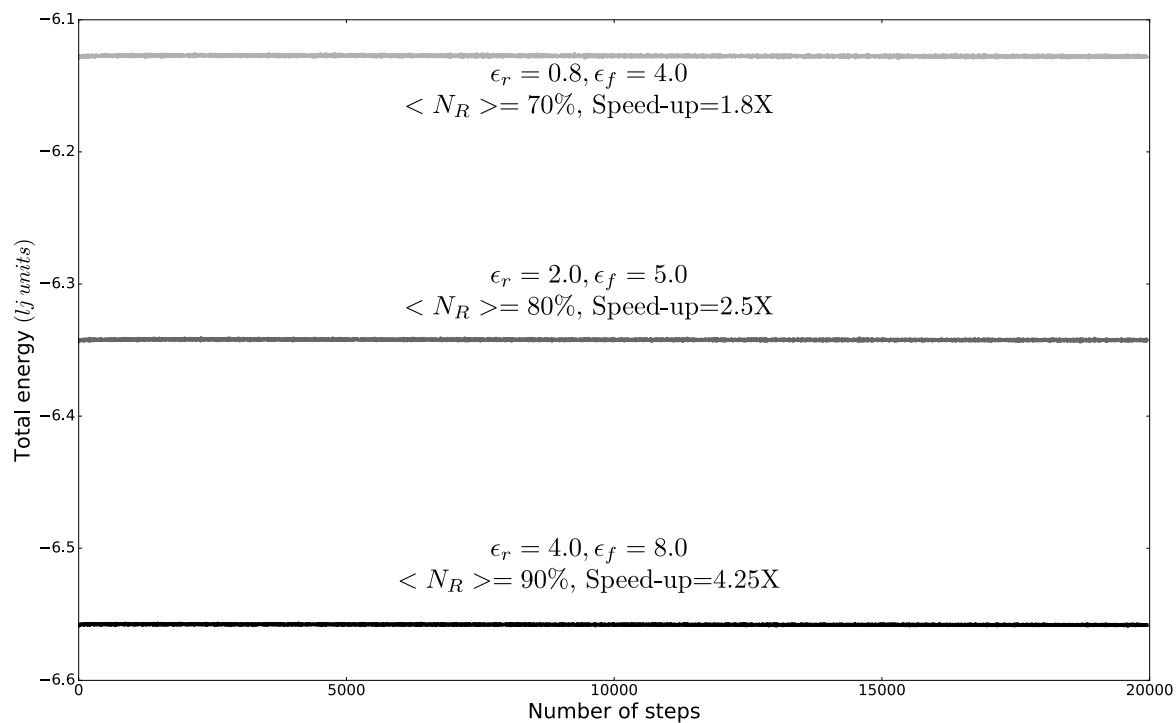
We performed this benchmark to show that AR molecular dynamics simulations in the NVE ensemble allow us to finely trade between precision and speed. In this benchmark, a high-velocity particle collides with an initially static 2D system containing 7290 particles. Interactions among particles are computed using a Lennard-Jones potential. We simulated this system for different values of AR parameters  $\epsilon_r$  and  $\epsilon_f$ , and various ratios  $\frac{\epsilon_f}{\epsilon_r}$ . All simulations were performed for 7000 steps with a time step size equal to 0.0003 (LJ units). We also performed a reference MD simulation of the same system using a Verlet integrator. To measure the deviation between ARMD simulations and the reference simulation, we extracted the last configuration of each ARMD simulation and computed the Root Mean Square Deviation (RMSD) with the last configuration of the reference simulation. In order to measure the speed-up achieved by AR simulations relatively to the classical simulation, we ran all simulations ten times and calculated the average computational cost of each time step.

Figure 4 illustrates the obtained speed-up as a function of AR parameters. We observed the highest speed-up (8.0 times) with  $\epsilon_f = 3*\epsilon_r$ . As the ratio of  $\epsilon_f/\epsilon_r$  increases, speed-up does not vary that much. For most values of AR parameters, we achieved 7 to

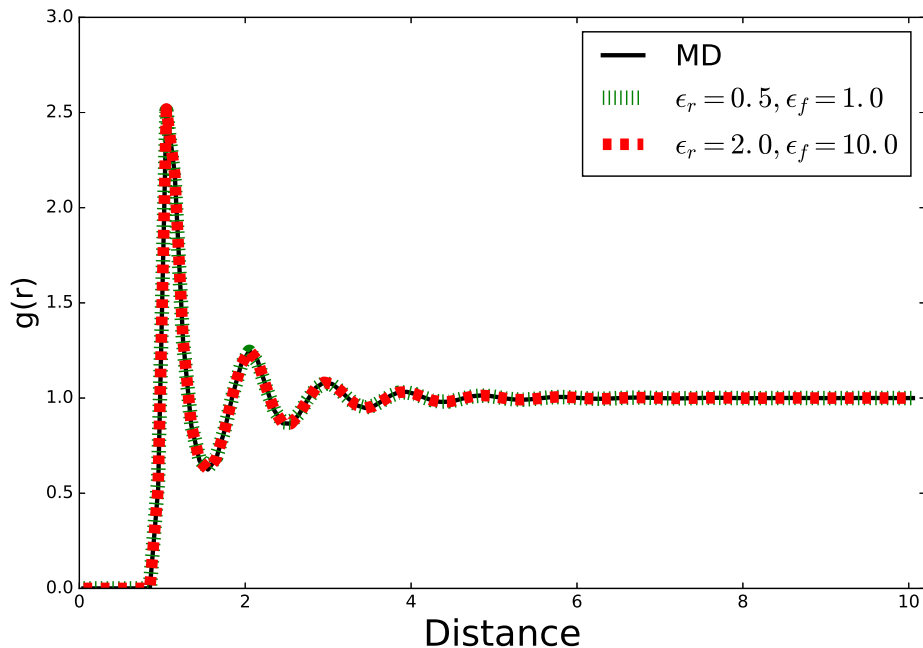
<sup>§</sup> Note that different AR parameters generate different Hamiltonians, so that identical initial conditions result in different total adaptive energies.



**Figure 1.** Speed-up achieved with ARMD as a function of the percentage of restrained particles.



**Figure 2.** ARMD simulation of 108,000 LJ particles with different AR parameters. The total energy remain constant irrespective of the AR parameters.



**Figure 3.** AR molecular dynamics preserves the radial distribution function in the NVT ensemble (108,000 LJ particles).

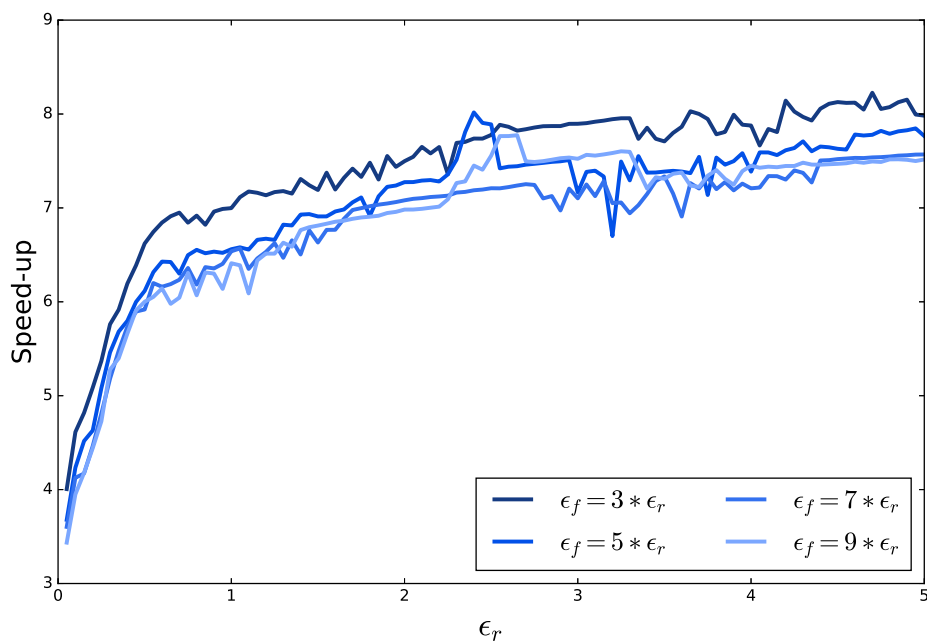
8 times speed-up. One reason for the smaller speed-up when the ratio between  $\epsilon_f$  and  $\epsilon_r$  increases is the larger number of particles belonging to the transition region, since updating positions of transitioning particles is more computationally involved than for particles belonging to other regions.

Figure 5 shows how ARMD deviates from standard MD when the AR parameters vary. From figure 5, we can infer that different AR parameters might result in the same RMSD, and that the relationship between the RMSD and the AR parameters may sometimes be non-trivial (such as the RMSD behavior when  $\epsilon_f = 3 * \epsilon_r$ ). Figure 6 represents the speed-up as a function of the average percentage of restrained particles. The speed-up is highly correlated with the average number of restrained particles, and thus to values of  $\epsilon_r$ . In this benchmark, we obtained an eight times speed-up when more than 98% of the particles were restrained, while still obtaining a RMSD from the reference simulation of  $0.07 \sigma$ .

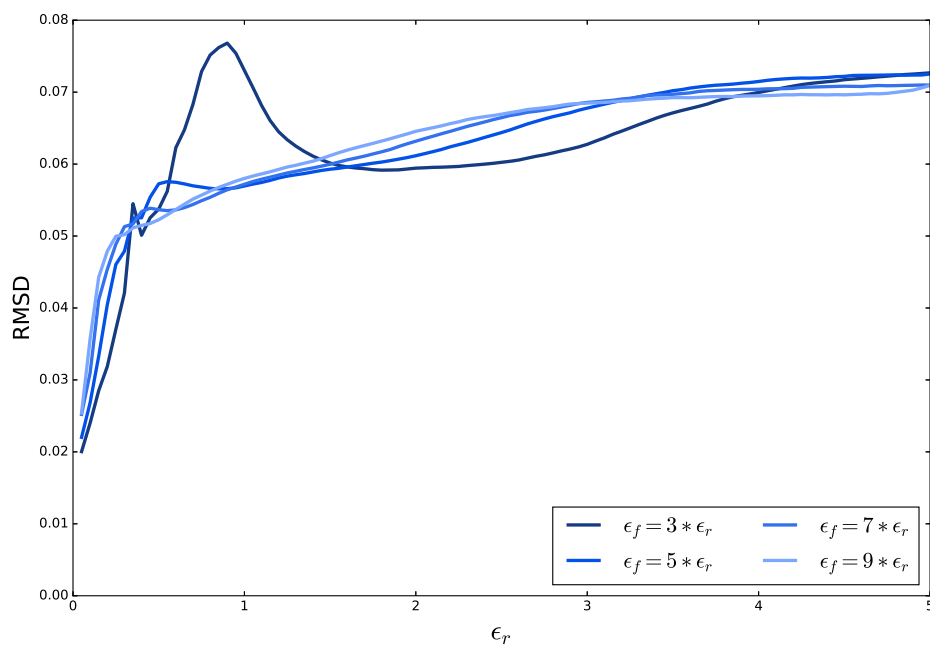
From this benchmark, it is evident that the achievable speed-up is related to the average number of restrained particles, which in turn is a function of  $\epsilon_r$ . Higher  $\epsilon_r$  and  $\epsilon_f$  values lead to higher speed-ups and, in general, higher RMSD values.

### 3.3. Toy model of an ion passing through a membrane channel

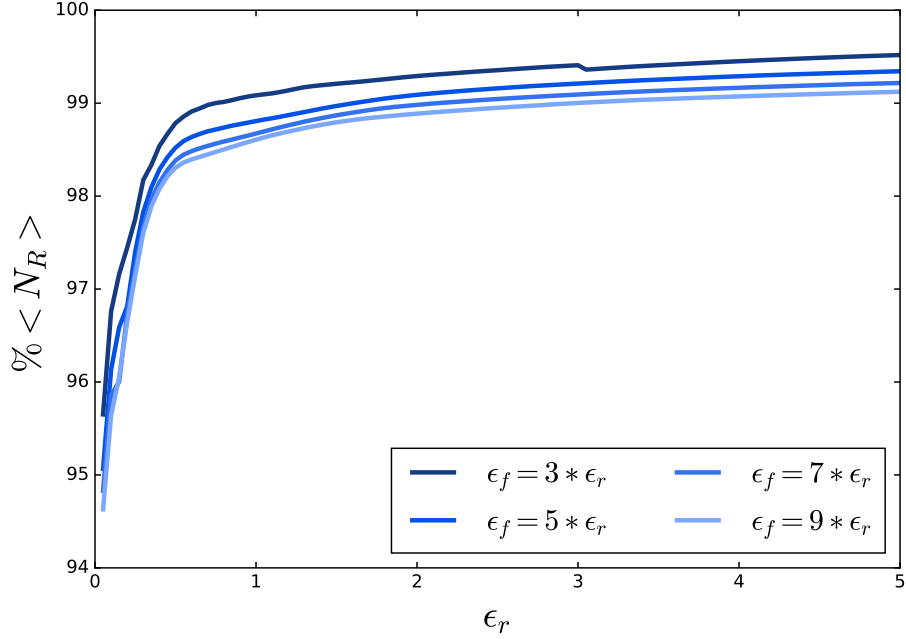
In order to mimic the biological process of an ion passing through a membrane channel, we created a 2D toy model. In biology, channels are usually proteins that are immersed in biological membranes. These proteins either have a channel or pore, or act as a gate



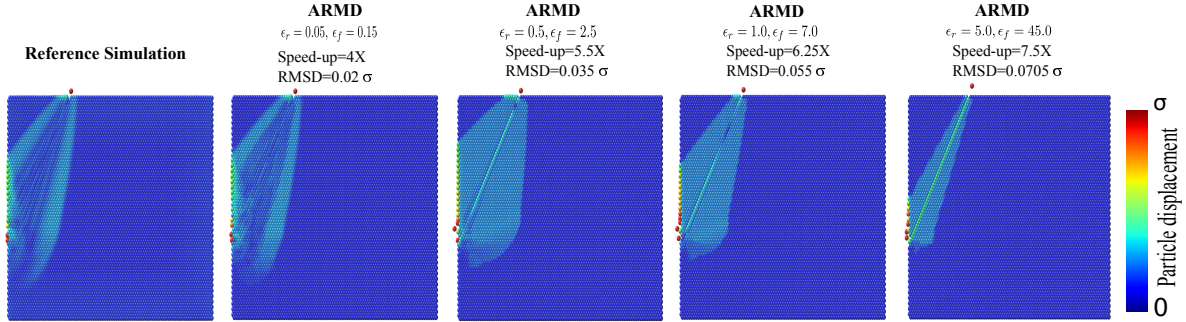
**Figure 4.** Collision cascade: speed-up with respect to the  $\epsilon_r$  values.



**Figure 5.** Collision Cascade: deviation from standard molecular dynamics with respect to the  $\epsilon_r$  values. Different colors indicate the relationship between  $\epsilon_r$  and  $\epsilon_f$  values.



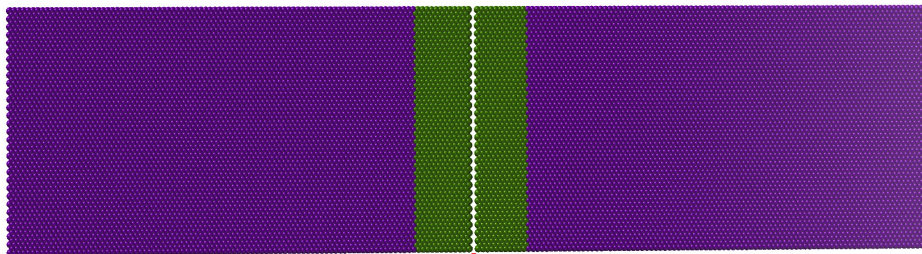
**Figure 6.** Collision Cascade: number of restrained particles as a function of  $\epsilon_r$  values.



**Figure 7.** Collision Cascade: ARMD simulations collision cascade of the same system with different AR parameters. Particles are colored according to the displacement from their initial positions. AR simulations in the NVE ensemble make it possible to finely trade between speed-up and precision.

controlling the passage of small molecules across the membrane. The movement of the small molecules is often driven by an electrochemical gradient across the membrane. In this toy model, the membrane and ions were represented with Lennard-Jones particles, and we applied an external force in the Y direction to model an electrochemical gradient across the membrane. Figure 8 represents the toy model. We divided the system into three types of particles:

- (i) Type 1: particles that represents an ion. The initial velocity of these particles is set in the Y direction only (red color particles in Figure 8).
- (ii) Type 2: channel particles, *i.e.* particles in close proximity to the passing ion (green particles in Figure 8).



**Figure 8.** Toy model of an ion passing through a channel (12,141 particles). This system contains 3 types of particles. The red particle is always active, green particles are less restrained compared to violet particles.

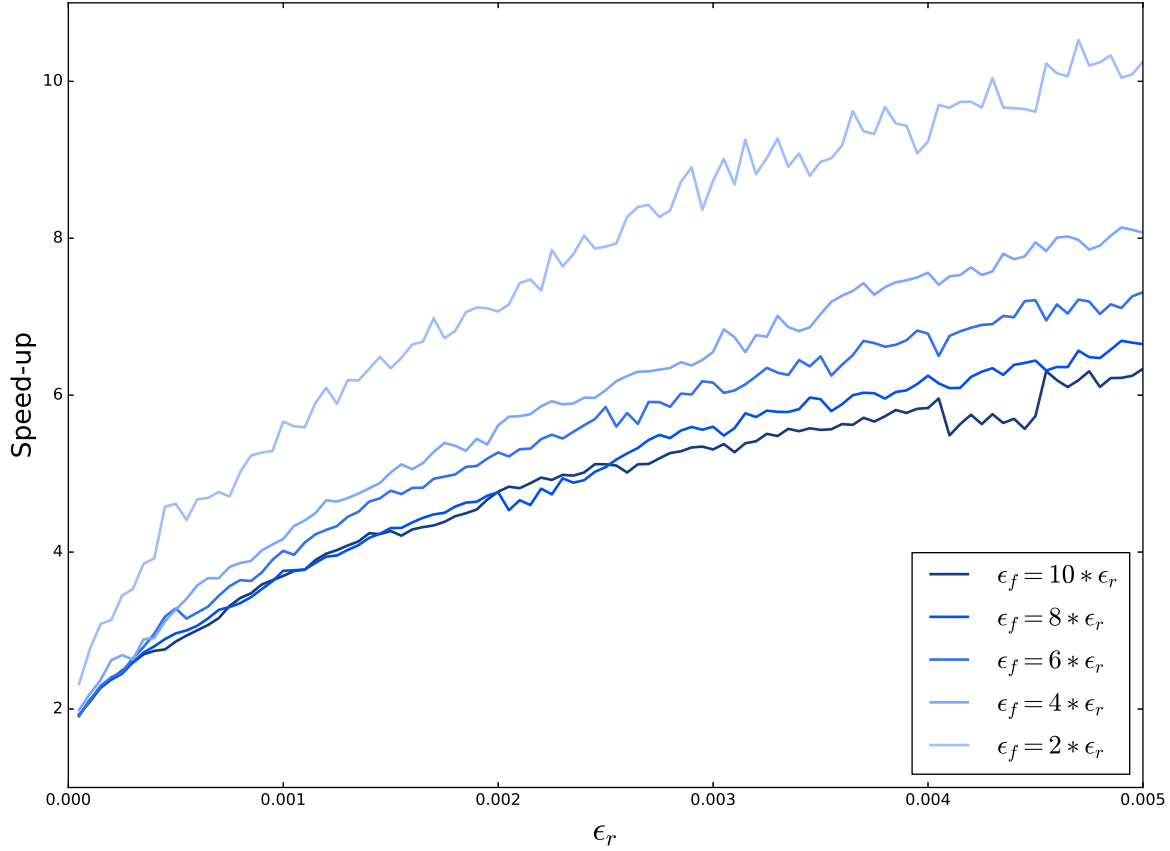
(iii) Type 3: membrane particles (violet particles in Figure 8).

In this system, containing 12,141 particles, Type 1 particles enter into the pore formed by Type 2 particles, and Type 2 particles either accelerate or decelerate Type 1 particles. Since we were interested in the motion of Type 1 particles (*e.g.* the speed at which they traverse the channel), we did not apply any restraint on these particles ( $\epsilon_r = \epsilon_f = 0$ ). For Type 2 particles, we set  $\epsilon_f$  to two, four, six, eight and ten times of  $\epsilon_r$  values. For Type 3 particles, AR parameters were five times larger than Type 2 particles. Each simulation was performed for 150,000 steps, with a time step equal to 0.0001.

In order to verify the properties of the system, we also ran a reference MD simulation using with a Verlet integrator. For speed-up measurement, we ran each simulation 10 times and computed the average time spent in each integration step. We measured the speed-up with respect to the reference simulation. We computed the probability density of type 1 particle along the Y direction (the channel axis) [27]. Figure 9 shows the obtained speed-up with respect to the  $\epsilon_r$  values of Type 2 particles. We achieved a 10X speed-up with  $\epsilon_f = 2 * \epsilon_r$ , and a 6-7X speed-up with other ratios of AR parameters. Higher speed-ups were observed with lower ratios of  $\epsilon_f/\epsilon_r$ , which we also observed in the collision cascade example. Figure 10 shows the probability density of an ion inside the toy system. From Figure 10, we can infer that, except for the largest ratio in AR parameters ( $\epsilon_f/\epsilon_r = 10$ ), all other simulations retain the same distribution as the reference simulation, hence the same ion circulation dynamics, while still allowing for large speed-ups.

### 3.4. Toy model of a solvated polymer

In order to demonstrate how ARMD may be used to estimate the statistical properties of a system in the NVT ensemble, we performed a simulation of a polymer in a cubical solvent box. The toy polymer contains a chain of 8 identical particles with mass 10 grams/mole. The solvent box (length 50 Å) contains 4,000 Lennard-Jones particles with mass 2.9 grams/mole. A harmonic potential was used for bonded interactions (bond, angle and dihedral terms), and a Lennard-Jones potential (cut-off 12.5 Å) was used for



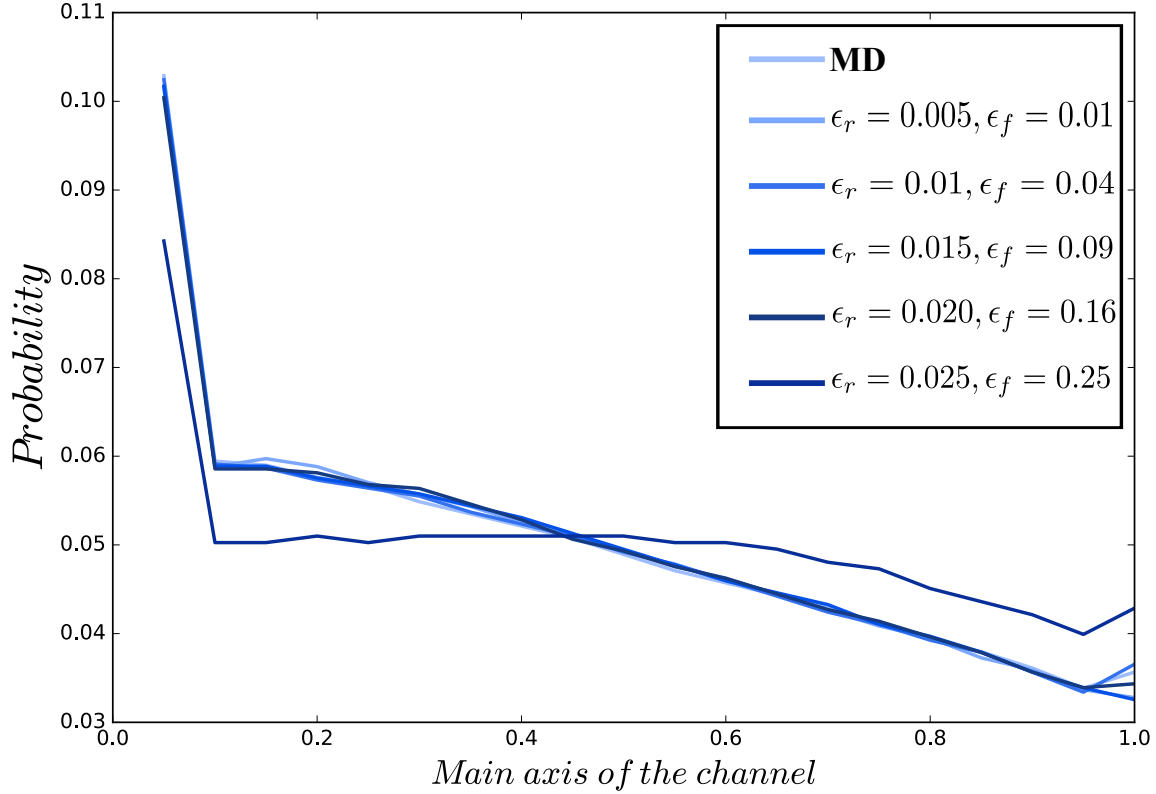
**Figure 9.** Toy model of an ion passing through a channel (12,141 particles). Speed-up as a function of the restrained parameter  $\epsilon_r$  for Type 2 particles.

non-bonded interactions. The system was initially minimized and then simulated in the NVT ensemble using a  $1fs$  time step. Initial velocities were generated using the Maxwell-Boltzmann distribution at a given temperature. Periodic boundary conditions were employed in all three directions. Since we wanted to compute statistic averages of the polymer, we did not apply any restraint on it ( $\epsilon_r = \epsilon_r = 0$ ), while solvent particles were restrained with  $\epsilon_r = 0.25Kcal/mol$  and  $\epsilon_f = 2.50Kcal/mol$ .

We simulated this system for different temperatures (300K, 400K, 500K, 600K and 700K). Figure 11 shows that the system takes few time steps to reach the desired temperature. For each temperature and combination of AR parameters, we observed an equilibrium in the number of active particles. The AR parameters also influence the time needed to reach this equilibrium and the desired temperature (Figures 12, 13, 14 and 15).

In order to verify statistical averages, we compared the radial distribution function (RDF) obtained by 10 *ns* of ARMD simulation with the one obtained by molecular dynamics, in the NVT ensemble at 300 K temperature, with the aforesaid parameters. On average, 36% of the particles were active during the ARMD simulation. Figure 16 shows that the RDF obtained by ARMD matches the one obtained with MD, indicating



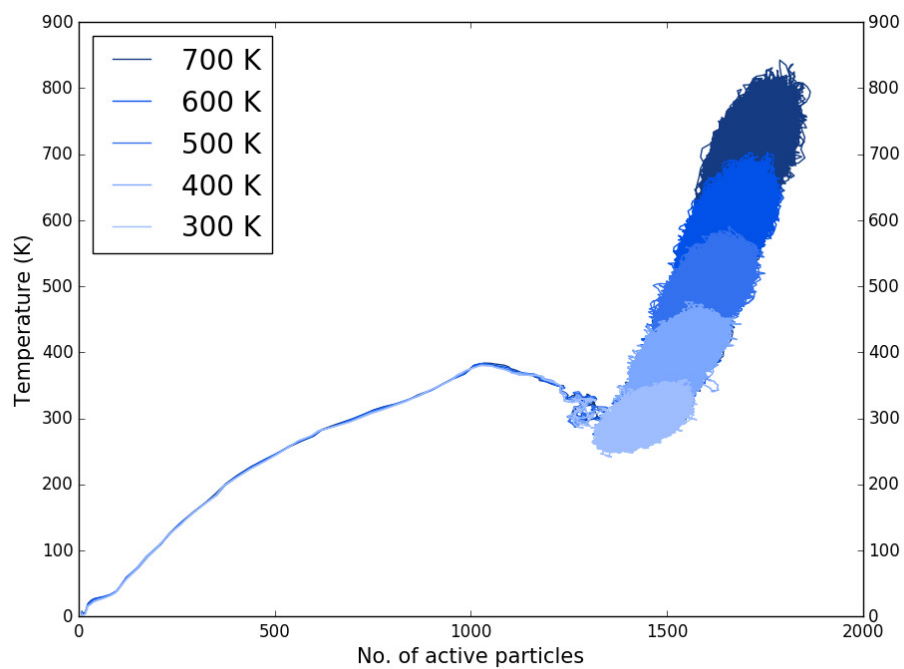


**Figure 10.** Toy model of an ion passing through a channel (12,141 particles). Differences in the ion probability density along the channel axis from the reference MD. Different plots represent different ratios of AR parameters.

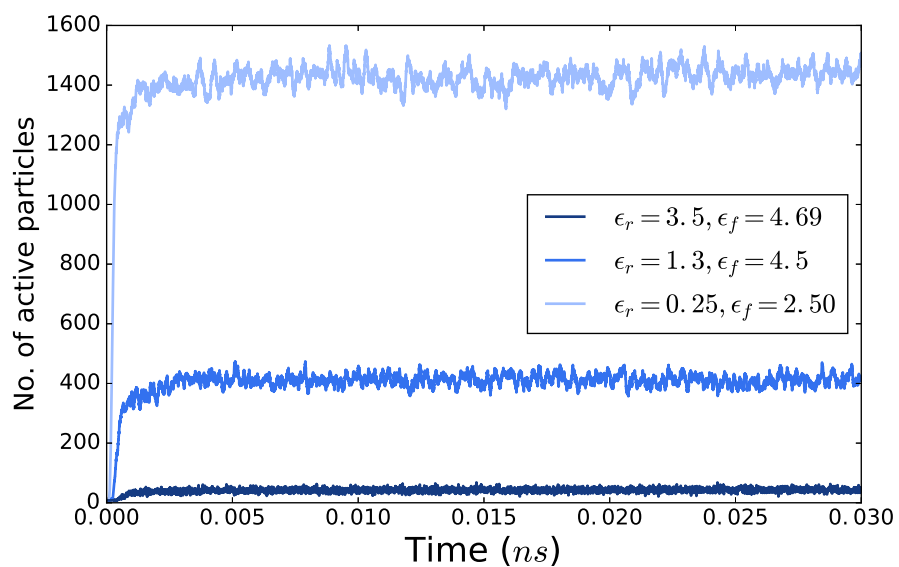
that position-dependent statistical averages are not modified [18].

In order to understand the effect of the number of active particles on temperature in ARMD, we performed three simulations with varying  $\epsilon_r$  and  $\epsilon_f$  values at 300K. Figure 17 shows that different  $\epsilon_r$  and  $\epsilon_f$  values lead to different numbers of active particles for the desired temperature. Figure 17 and table 1 illustrate how fluctuations in temperature are also related to the number of active particles, and fluctuations of the number of active particles: higher percentages of restrained particles lead to higher temperature fluctuations.

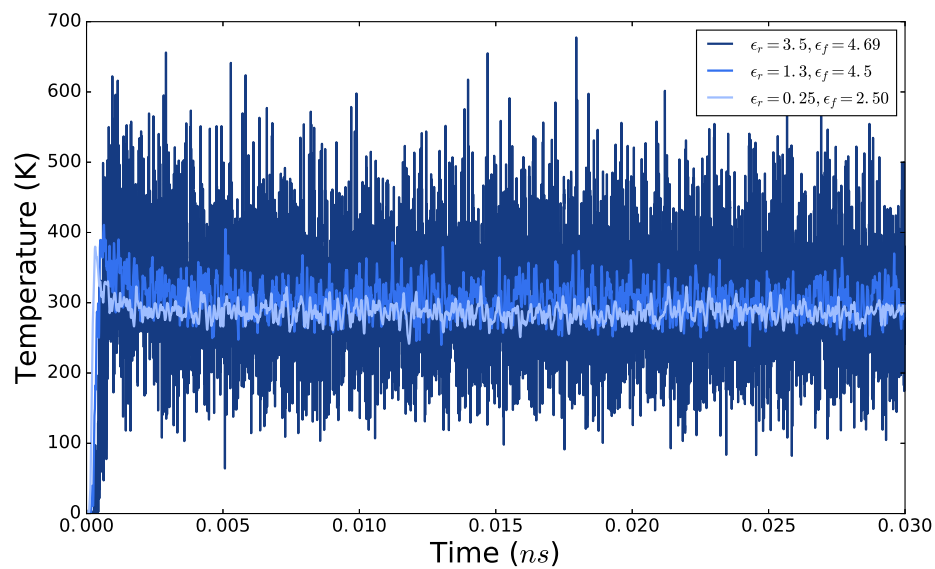
In the NVT ensemble, the overall achievable speed-up when computing a statistical average depends both on the instantaneous computational speed-up that can be achieved at each time step, and the modification of the variance caused by the AR Hamiltonian [25]. We refer the reader to Artemova and Redon [18] for an example analysis of the speed-up that can be achieved in the NVT ensemble.



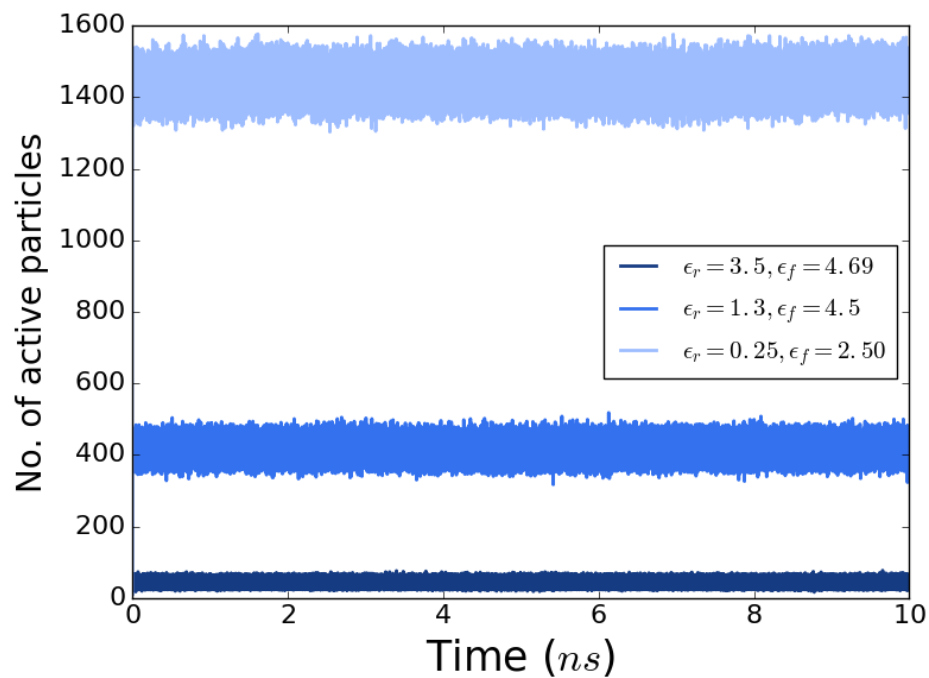
**Figure 11.** Variation of temperature with respect to the number of active particles.



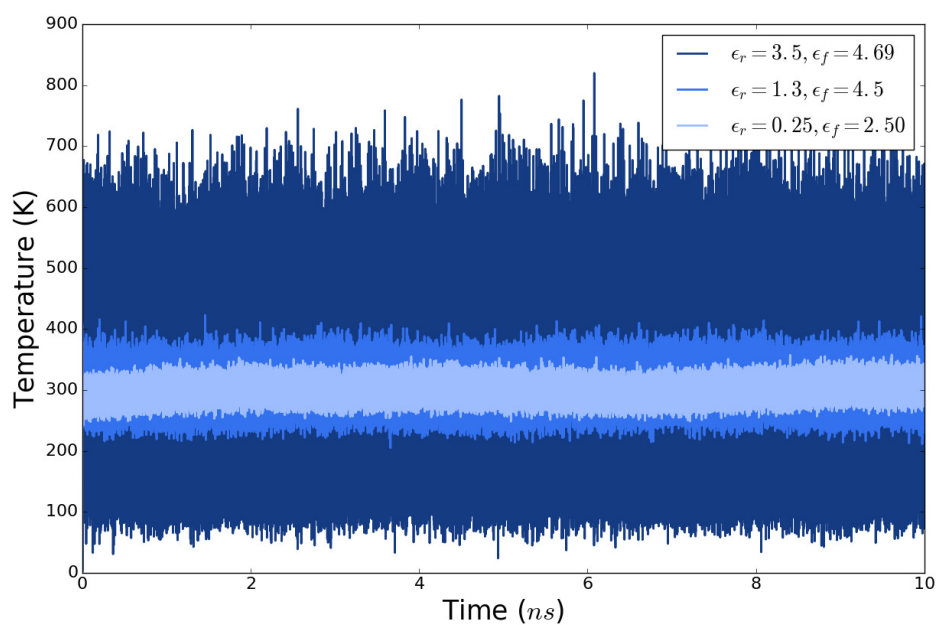
**Figure 12.** Equilibrium period for the number of active particles.



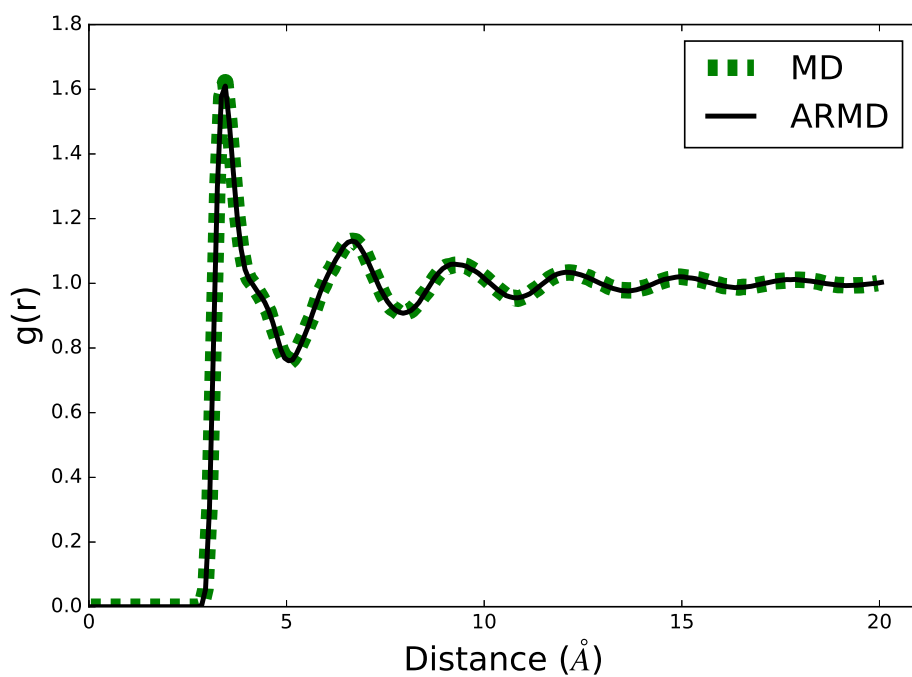
**Figure 13.** Equilibrium period for the instantaneous temperature.



**Figure 14.** Time evolution of the number of active particles.



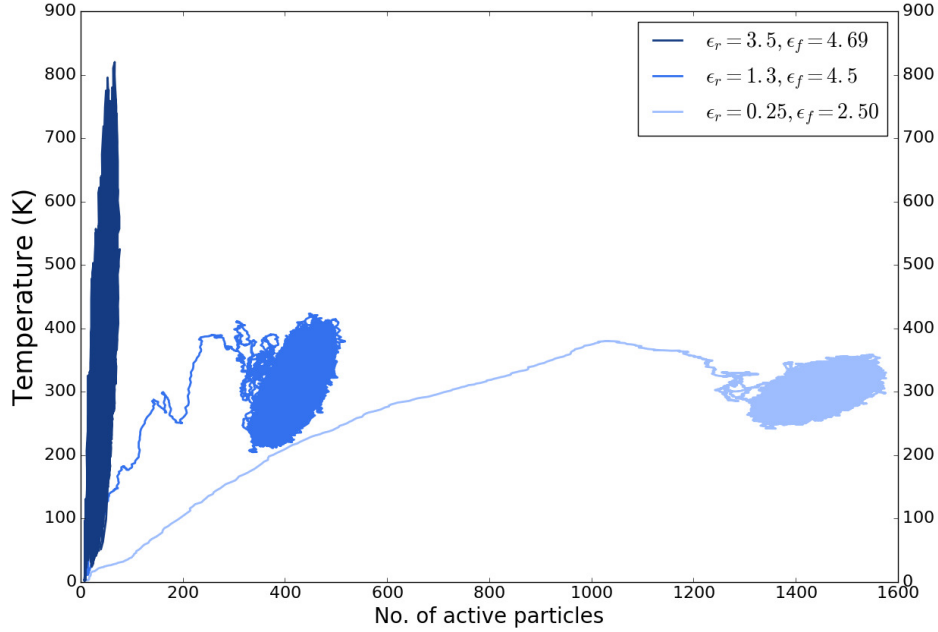
**Figure 15.** Instantaneous temperature of the system with different AR parameters.



**Figure 16.** Radial distribution functions of polymer obtained with ARMD and with MD

$\epsilon_r$	$\epsilon_f$	% $< N_{res} >$	avg. Temp. (k)	St.dev.
0.25	2.50	63.5	301.94	12.302217
1.3	4.5	90.0	299.501	22.913903
3.5	4.69	98.8	302.456	78.788141

**Table 1.** Table represents the average and standard deviation in temperature obtained by 3 different ARMD simulations with different AR parameters.



**Figure 17.** Temperature profile of the system with respect to the number of active particles with variation in  $\epsilon_r$  and  $\epsilon_f$  values.

#### 4. Conclusion

We have presented novel algorithms enabling the use of AR molecular dynamics simulations with the LAMMPS software package, and we have demonstrated how ARMD makes it possible to speed up simulations.

We would now like to investigate extensions of this work in several directions. First, the current method needs to incrementally update forces twice (to subtract old contributions and then add new contributions). As a result, with this force update algorithm, ARMD requires about at least 60% restrained particles to achieve a speed-up. We thus need to explore the possibility of developing other force update algorithms. Second, we have described serial algorithms, even though LAMMPS may take advantage of parallelization (*e.g.* OpenMP and MPI) to speed up calculations for large systems. We need to extend our algorithms to this case (and potentially take care of load balancing issues). We also need to extend our force update algorithms to long-range interactions. Finally, we want to apply our methods to processes that we believe should strongly benefit from the ARMD methodology, *i.e.* systems where precise information should be

obtained on a specific part of the system.

## Acknowledgments

We gratefully acknowledge funding from Rhone-Alpes Region through the ARC program, and the European Research Council through the ERC Starting Grant n. 307629.

## References

- [1] Masakazu Matsumoto, Shinji Saito, and Iwao Ohmine. Molecular dynamics simulation of the ice nucleation and growth process leading to water freezing. *Nature*, 416(6879):409–413, March 2002. 00529.
- [2] F. Benkabou, H. Aourag, and M. Certier. Atomistic study of zinc-blende CdS, CdSe, ZnS, and ZnSe from molecular dynamics. *Materials Chemistry and Physics*, 66(1):10–16, September 2000. 00059.
- [3] W. A. Curtin and Ronald E. Miller. Atomistic/continuum coupling in computational materials science. *Modelling and Simulation in Materials Science and Engineering*, 11(3):R33, 2003. 00519.
- [4] Christopher A. Schuh and Alan C. Lund. Atomistic basis for the plastic yield criterion of metallic glass. *Nature Materials*, 2(7):449–452, July 2003. 00317.
- [5] Wilfred F. van Gunsteren and Herman J. C. Berendsen. Computer Simulation of Molecular Dynamics: Methodology, Applications, and Perspectives in Chemistry. *Angewandte Chemie International Edition in English*, 29(9):992–1023, September 1990. 01411.
- [6] R. Elber and M. Karplus. Multiple conformational states of proteins: a molecular dynamics analysis of myoglobin. *Science*, 235(4786):318–321, January 1987. 00665.
- [7] Michael Levitt. Molecular dynamics of native protein. *Journal of Molecular Biology*, 168(3):595–617, August 1983. 00373.
- [8] Andrea Amadei, Antonius B. M. Linssen, and Herman J. C. Berendsen. Essential dynamics of proteins. *Proteins: Structure, Function, and Bioinformatics*, 17(4):412–425, December 1993. 02006.
- [9] Martin Karplus and J. Andrew McCammon. Molecular dynamics simulations of biomolecules. *Nature Structural & Molecular Biology*, 9(9):646–652, September 2002. 01727.
- [10] Donald Hamelberg, John Mongan, and J. Andrew McCammon. Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules. *The Journal of Chemical Physics*, 120(24):11919–11929, June 2004. 00592.
- [11] Alessandro Laio and Michele Parrinello. Escaping free-energy minima. *Proceedings of the National Academy of Sciences*, 99(20):12562–12566, October 2002. 02189.
- [12] Darryl D. Humphreys, Richard A. Friesner, and Bruce J. Berne. A Multiple-Time-Step Molecular Dynamics Algorithm for Macromolecules. *The Journal of Physical Chemistry*, 98(27):6885–6892, July 1994. 00170.
- [13] W. B. Streett, D. J. Tildesley, and G. Saville. Multiple time-step methods in molecular dynamics. *Molecular Physics*, 35(3):639–648, March 1978. 00257.
- [14] Peter J. Bond, John Holyoake, Anthony Ivetac, Syma Khalid, and Mark S. P. Sansom. Coarse-grained molecular dynamics simulations of membrane proteins and peptides. *Journal of Structural Biology*, 157(3):593–605, March 2007. 00241.
- [15] Cecilia Clementi. Coarse-grained models of protein folding: toy models or predictive tools? *Current Opinion in Structural Biology*, 18(1):10–15, February 2008. 00225.
- [16] Valentina Tozzini. Coarse-grained models for proteins. *Current Opinion in Structural Biology*, 15(2):144–150, April 2005. 00652.

- [17] David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J. P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang. Anton, a Special-purpose Machine for Molecular Dynamics Simulation. *Commun. ACM*, 51(7):91–97, July 2008. 00358.
- [18] Svetlana Artemova and Stephane Redon. Adaptively Restrained Particle Simulations. *Physical Review Letters*, 109(19):190201, November 2012. 00008.
- [19] Mael Bosson, Sergei Grudin, Xavier Bouju, and Stephane Redon. Interactive physically-based structural modeling of hydrocarbon systems. *Journal of Computational Physics*, 231(6):2581–2598, March 2012. 00015.
- [20] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1–19, March 1995. 10760.
- [21] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh Ewald: An N.log(N) method for Ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, June 1993. 11098.
- [22] Pedro Gonnet. A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. *Journal of Computational Chemistry*, 28(2):570–573, January 2007. 00046.
- [23] G. Sutmann and V. Stegailov. Optimization of neighbor list techniques in liquid matter simulations. *Journal of Molecular Liquids*, 125(2–3):197–203, April 2006. 00036.
- [24] Eduard S. Fomin. Consideration of data load time on modern processors for the Verlet table and linked-cell algorithms. *Journal of Computational Chemistry*, 32(7):1386–1399, May 2011. 00023.
- [25] Zofia Trstanova and Stephane Redon. Estimating the speed-up for Adaptively Restrained Langevin Dynamics. *Submitted*.
- [26] Stephane Redon, Gabriel Stoltz, and Zofia Trstanova. Error Analysis of Modified Langevin Dynamics. *Journal of Statistical Physics*, pages 1–37, June 2016. 00000.
- [27] Sharron Bransburg-Zabary, Esther Nachliel, and Menachem Gutman. A Fast in Silico Simulation of Ion Flux through the Large-Pore Channel Proteins. *Biophysical Journal*, 83(6):3001–3011, December 2002. 00008.